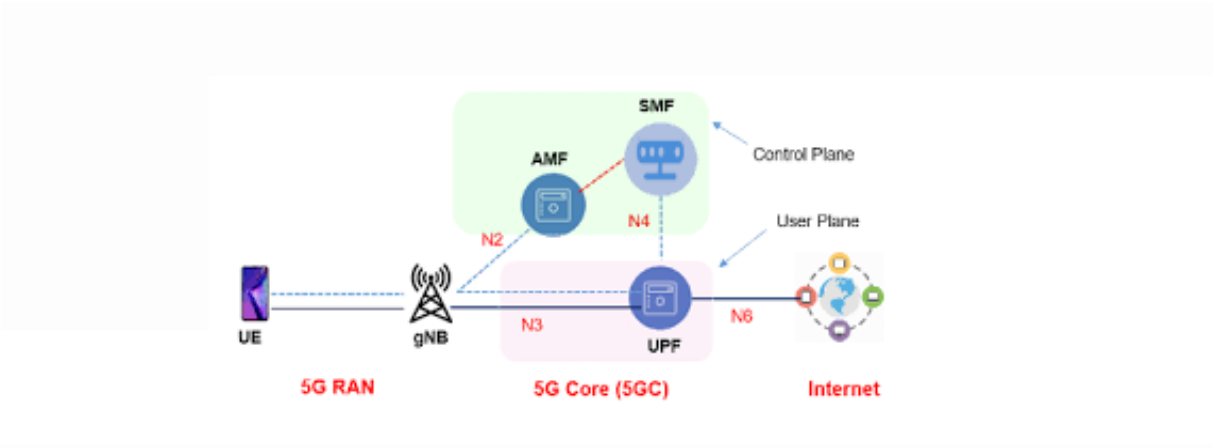
Wireless Networks

5G Wireless Network (Lab)

Demonstrate distributed UPF deployment and slice-base UPF selection

5G network Architecture Diagram



ComNetsEmu

ComNetsEmu is a testbed and network emulator designed for the NFV/SDN teaching book "Computing in Communication Networks: From Theory to Practice". The design focuses on emulating all examples and applications on a single computer, for example on a laptop. ComNetsEmu extends the famous Mininet network emulator to support better emulation of versatile **Computing In The Network (COIN) applications**. It extends and puts forward the concepts and work in the Containernet project. It uses a slightly different approach to extend the Mininet compared to Containernet. It's main focus is to use "sibling containers" to emulate network systems with **computing**. See a more detailed comparison between upstream Mininet and Containernet here.

Main Features

- Use Docker hosts in Mininet topologies.
- Manage application Docker containers deployed **inside** Docker hosts. "Docker-in-Docker" (sibling containers) is used as a lightweight emulation of nested virtualization. A Docker host with multiple **internal** Docker containers deployed is used to **mimic** an actual physical host running Docker containers (application containers).
- A collection of application examples for "Computing In Communication Networks" with sample codes and detailed documentation. All examples can be easily reproduced and extended.

Check the Roadmap for planed and WIP features.

Installation

It is highly recommended to run ComNetsEmu **inside** a virtual machine (VM). **Root privileges** are required to run the ComNetsEmu/Mininet applications.

ComNetsEmu runs smoothly in a VM with 2 vCPUs and 2GB RAM. (Host Physical CPU: Intel i7-7600U @ 2.80GHz)

There are some options to install ComNetsEmu

Option 1: Install in a Vagrant managed VM (Highly Recommended)

Option 2: Install on user's custom VM or directly on host OS

Option 3: Download the pre-built VM image

Link https://stevelorenz.github.io/comnetsemu/installation.html#option-1-install-in-a-vagrant-managed-vm-highly-recommended

We will install with option 1

Prerequisite

- Vagrant: >= v2.2.5 (Download Link)
  Vagrant is the command line utility for managing the lifecycle of virtual machines.
- Virtualbox: >= v6.0 (Download Link)

  VirtualBox is a free and open-source virtualization software that allows users to create and run virtual machines on their compute

Some Known Issues

Please follow the documentation to setup the VM with proper resource allocation. If the `vagrant up` command fails to setup the VM fully correct which you can test by running the basic *Docker-in-Docker* example. Please firstly check the known VM setup issues for potential solutions.

VM Resource Allocation (**Important**)

By default, this VM is allocated with **2** vCPUs and **4GB** RAM to run all examples and applications smoothly. If you machine does not have enough resources, you need to change the variable *CPUS* and *RAM* in the Vagrantfile **before** created the VM.

PART 1

Let's install

```
$ cd ~
$ git clone https://git.comnets.net/public-repo/comnetsemu.git

$ cd ./comnetsemu


$ vagrant up comnetsemu

# This will create the VM at the first time (takes around 20 minutes)
$ vagrant up comnetsemu

# SSH into the VM
$ vagrant ssh comnetsemu (default username and password is vagrant)

# Power off the VM
$ vagrant halt comnetsemu

# Remove/Delete the VM
$ vagrant destroy comnetsemu
```

If you see

Congratulations! The installation is done successfully!

For users running Windows as the host OS:

Warning: Main developers of ComNetsEmu does not use Windows and does not have a Windows machine to test on.

1. If you are using Windows, we recommend using Mobaxterm as the console. This should solve problems opening `xterm` in the emulator. It is also a SSH client for Windows that provides support for X11 forwarding, which is required for running GUI applications in a remote Linux environment.

After installing mobaXterm lets ssh

1. Download and install MobaXterm from the official website: https://mobaxterm.mobatek.net/download.html
2. Open MobaXterm and click on "Session" in the top left corner.
3. In the "Session settings" window, select "SSH" as the protocol go into advance ssh setting and enable x11-farwording
4. In the "Remote host" field, enter "127.0.0.1".
5. In the "Specify port" field, enter "2222".
6. In the "Advanced SSH settings" section, select "Use internal SSH agent" and "Use private key" (if you have a private key set up for the VM).
7. In the "Specify username" field, enter "vagrant".
8. In the "Specify password" field, enter "vagrant".

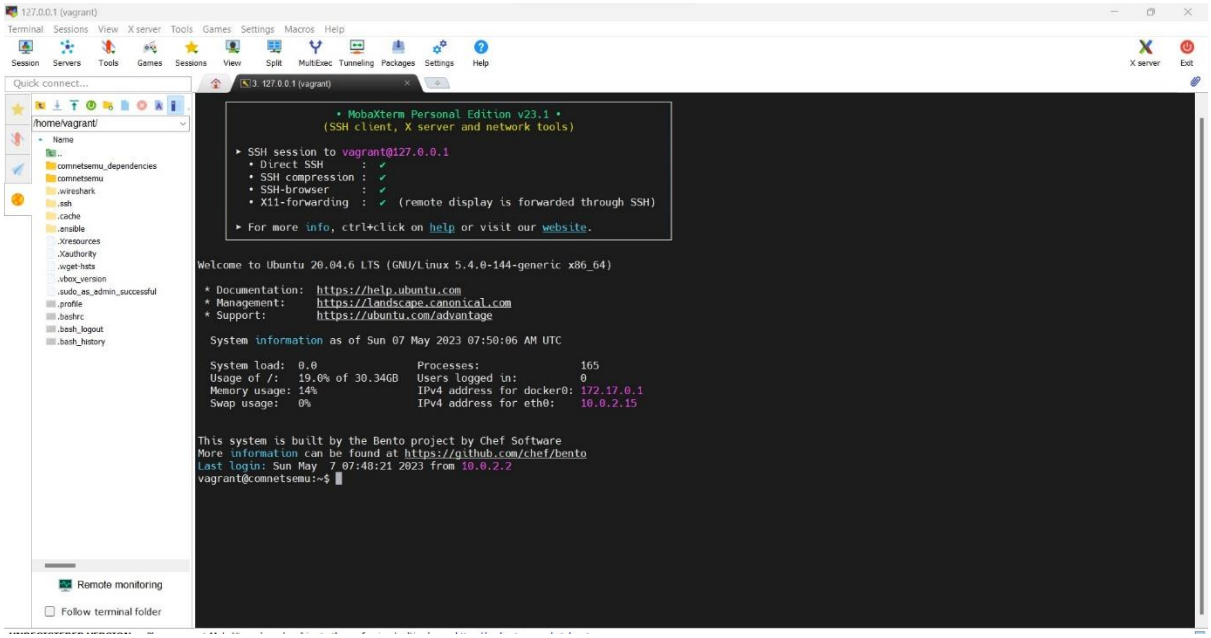
9. Click on "OK" to save the session settings.
10. Double-click on the session to connect to the VM via SSH using MobaXterm.

Note: Make sure that the Vagrant VM is running and that SSH is enabled on port 2222 in the Vagrantfile.

Please note If you see

This system is built by the Bento project by Chef Software More information can be found at https://github.com/chef/bento /usr/bin/xauth: file /home/vagrant/.Xauthority does not exist

Then To resolve this issue, you can try restarting the X server by logging out and logging back in or by restarting the virtual machine



Now lets upgrade

Step 1: Upgrade source code of ComNetsEmu Python package, examples and applications

Use git to pull (or fetch+merge) the latest tag (or commit) in master branch:

```
$ cd ~/comnetsemu
$ git checkout master
$ git pull origin master
```

Step 2: Automatically upgrade ComNetsEmu Python modules and all dependencies

The installer script is used to perform this step. Run following commands **inside** the VM to upgrade automatically:

```
$ cd ~/comnetsemu/util
$ bash ./install.sh -u
```

The script may ask you to input yes or no several times, please read the terminal output for information.

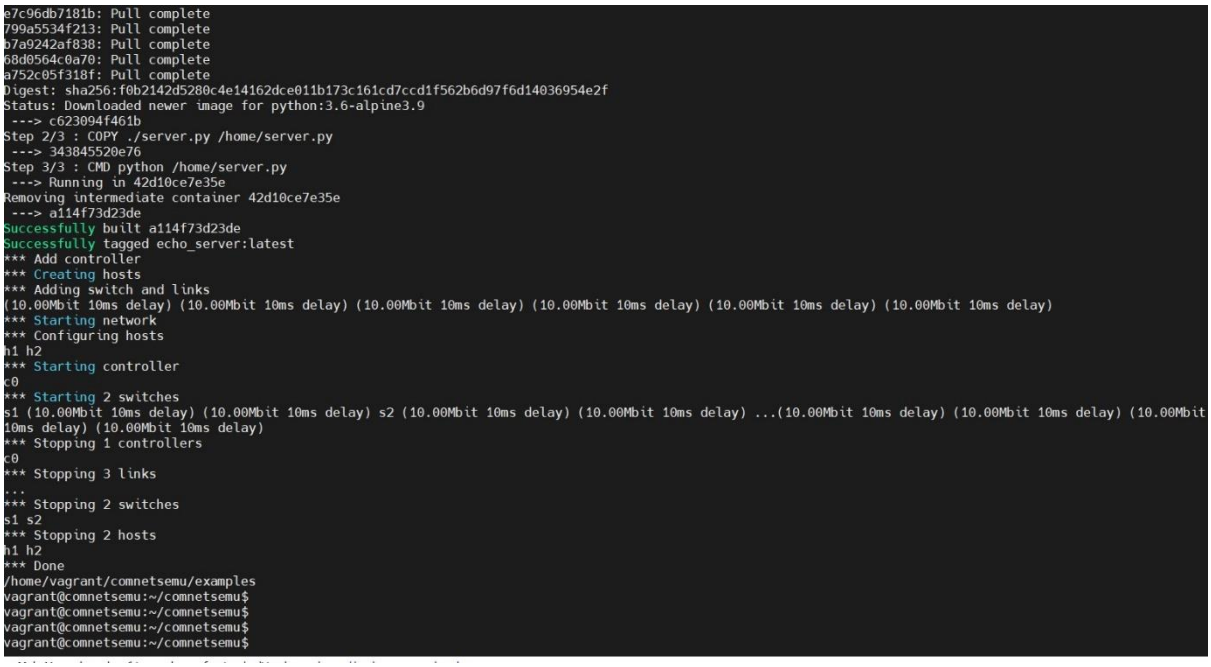
When finish you will see results like this

Step 3: Check if the upgrade is successful

Run following commands inside the VM to run tests:

```
$ cd ~/comnetsemu/
$ sudo make test && sudo make test-examples
```

If all tests pass without any errors or exceptions, the upgrading was successful.



If you don’t see this it is recommended to redo the upgrade process or just rebuild the Vagrant VM if the situation is too bad...

PART 2

Know lets Emulate a 5G network deployment in comnetsemu. Demonstrate distributed UPF deployment and slice-base UPF selection.

We will use Riccardo Fedrizzi deployment

https://github.com/RiccardoFedrizzi/comnetsemu_5Gnet

Steps:

cd comnetsemu/app

git clone https://github.com/RiccardoFedrizzi/comnetsemu_5Gnet.git

cd comnetsemu_5Gnet/build

```
./build.sh
```

Or alternatively download them from DockerHub

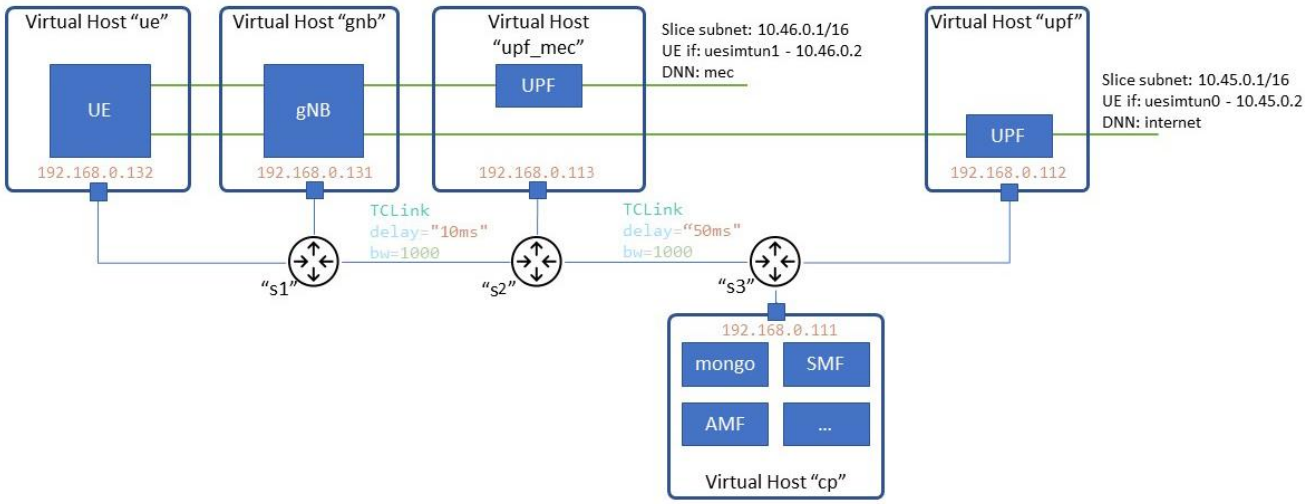
./dockerhub_pull.sh

If you see

```
Setting up iperf3 (3.7-3) ...
Setting up iptables (1.8.4-3ubuntu2) ...
update-alternatives: using /usr/sbin/iptables-legacy to provide /usr/sbin/iptables (iptables) in auto mode
update-alternatives: using /usr/sbin/ip6tables-legacy to provide /usr/sbin/ip6tables (ip6tables) in auto mode
update-alternatives: using /usr/sbin/arptables-nft to provide /usr/sbin/arptables (arptables) in auto mode
update-alternatives: using /usr/sbin/ebtables-nft to provide /usr/sbin/ebtables (ebtables) in auto mode
Setting up libhx509-5-heimdal:amd64 (7.7.0+dfsg-1ubuntu1.4) ...
Setting up libsctp-dev:amd64 (1.0.18+dfsg-1) ...
Setting up libkrb5-26-heimdal:amd64 (7.7.0+dfsg-1ubuntu1.4) ...
Setting up libheimntlm0-heimdal:amd64 (7.7.0+dfsg-1ubuntu1.4) ...
Setting up libgssapi3-heimdal:amd64 (7.7.0+dfsg-1ubuntu1.4) ...
Setting up libldap-2.4-2:amd64 (2.4.49+dfsg-2ubuntu1.9) ...
Setting up libcurl4:amd64 (7.68.0-1ubuntu2.18) ...
Setting up curl (7.68.0-1ubuntu2.18) ...
Processing triggers for libc-bin (2.31-0ubuntu9.9) ...
Reading package lists...
Building dependency tree...
Reading state information...
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Reading package lists...
Building dependency tree...
Reading state information...
Removing intermediate container f2487cdce4c6
 ---> 8fe2b63c4142
Step 9/11 : COPY --from=builder /UERANSIM/build /UERANSIM/build
 ---> 0d8989582c33
Step 10/11 : COPY --from=builder /UERANSIM/config /UERANSIM/config
 ---> 056221790aaa
Step 11/11 : WORKDIR /UERANSIM/build
 ---> Running in 0b50d094b024
Removing intermediate container 0b50d094b024
 ---> 9eb70b100f99
Successfully built 9eb70b100f99
Successfully tagged myueransim_v3-2-6:latest
vagrant@comnetsemu:~/comnetsemu/app/comnetsemu_5Gnet/build$
vagrant@comnetsemu:~/comnetsemu/app/comnetsemu_5Gnet/build$
```

Congratulations! we have installed 5G network deployment in comnetsemu. Demonstrate distributed UPF deployment and slice-base UPF selection

let’s understand our network Topology

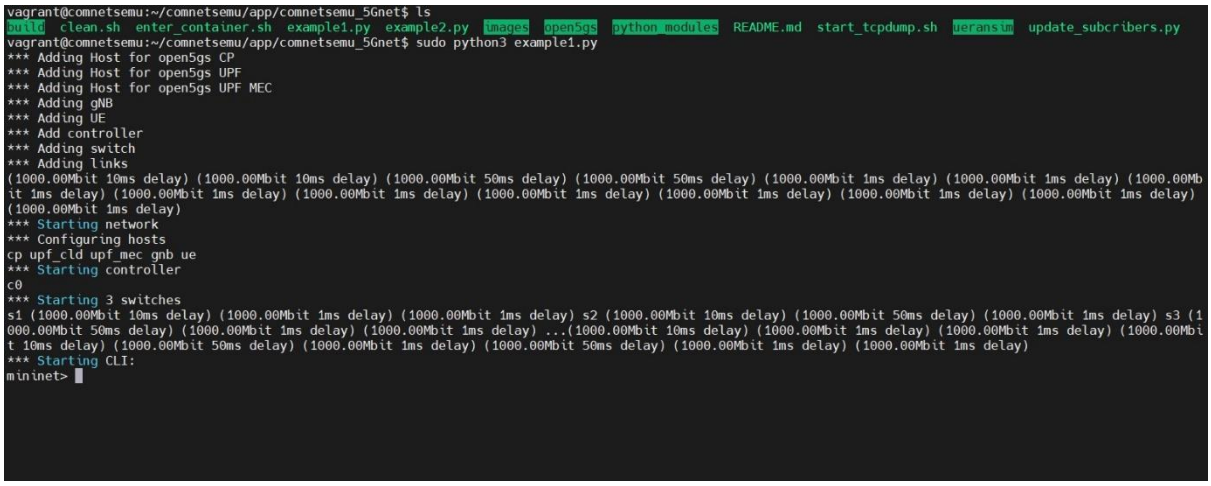


Run experiments

Start the network topology:

Running example1.py

```
$ sudo python3 example1.py
```



Some commands for Mininet that you can use to analyse network configuration

1. `net` - Display the current network configuration.
2. `nodes` - List all nodes in the network.
3. `links` - List all links in the network.
4. `dump` - Dump the network configuration to a string.
5. `pingall` - Ping between all hosts in the network.
6. `intfs` - List all interfaces in the network.

Now if you want to add UE manually you will need to install browser

Install Falkon using apt.

Open a Terminal window and run the following command:

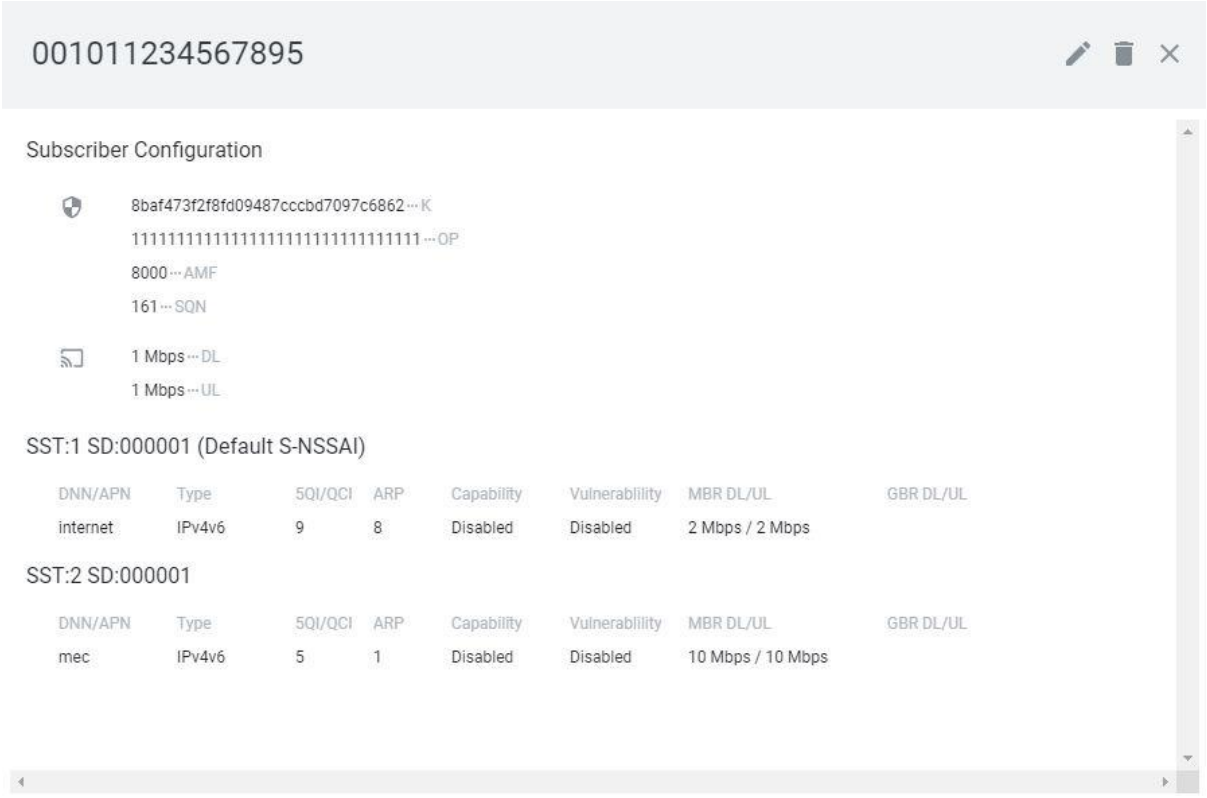
```
sudo apt update && sudo apt install falkon
```

after this try to run reboot

sudo reboot now

now run falkon and go to

http://127.0.0.1:3000 to access webUI and then add UE



Subscriber configuration should be like above diagram.

or we can do it by running example2.py

Running example2.py

This example creates the same environment of example1.py but the open5GS control plane configuration is done programmatically without using the webUI. (Note: adapted the python class in the open5gs repo here)

Disclaimer: all the previous subcribers registered with the webUI will be lost and a new one will be created.

```
$ sudo python3 example2.py
```

But before running example2 we need to install pymongo

sudo pip install pymongo

Check UE connections

Notice how the UE DockerHost has been initiated running `open5gs_ue_init.sh` which, based on the configuration provided in `open5gs-ue.yaml`, creates two default UE connections. The sessions are started specifying the slice, not the APN. The APN, and thus the associated UPF, is selected by the 5GC since, in `subscriber_profile.json`, a slice is associated to a session with specific DNN.
Enter the container and verify UE connections:

Open new session and then

cd comnetsemu/app/comnetsemu_5Gnet/

```
./enter_container.sh ue
ifconfig
```

```
root@27a0b846248b:/UERANSIM/build# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.6  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:06  txqueuelen 0  (Ethernet)
        RX packets 27  bytes 3950 (3.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ue-s1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.132  netmask 255.255.255.0  broadcast 0.0.0.0
        ether 3a:67:96:af:4c:6c  txqueuelen 1000  (Ethernet)
        RX packets 59  bytes 7330 (7.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12  bytes 1045 (1.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

uesimtun0: flags=369<UP,POINTOPOINT,NOTRAILERS,RUNNING,PROMISC>  mtu 1400
        inet 10.45.0.2  netmask 255.255.255.255  destination 10.45.0.2
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 500  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

uesimtun1: flags=369<UP,POINTOPOINT,NOTRAILERS,RUNNING,PROMISC>  mtu 1400
        inet 10.46.0.2  netmask 255.255.255.255  destination 10.46.0.2
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 500  (UNSPEC)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@27a0b846248b:/UERANSIM/build#
```

Start a ping test to check connectivity:

```
# ping -c 3 -n -I uesimtun0 www.google.com
```

```
root@c5736900920c:/UERANSIM/build# ping -c 3 -n -I uesimtun0 www.google.com
PING www.google.com (142.250.180.132) from 10.45.0.2 uesimtun0: 56(84) bytes of data.
64 bytes from 142.250.180.132: icmp_seq=1 ttl=107 time=194 ms
64 bytes from 142.250.180.132: icmp_seq=2 ttl=107 time=211 ms
64 bytes from 142.250.180.132: icmp_seq=3 ttl=107 time=190 ms
```

```
# ping -c 3 -n -I uesimtun1 www.google.com
```

```
root@c5736900920c:/UERANSIM/build# ping -c 3 -n -I uesimtun1 www.google.com
PING www.google.com (142.250.180.132) from 10.46.0.2 uesimtun1: 56(84) bytes of data.
64 bytes from 142.250.180.132: icmp_seq=1 ttl=107 time=219 ms
64 bytes from 142.250.180.132: icmp_seq=2 ttl=107 time=239 ms
64 bytes from 142.250.180.132: icmp_seq=3 ttl=107 time=103 ms
```

Test the environment

In two terminals start two tcpdump for both upf_cld and upf_mec

```
$ bash ./start_tcpdump.sh upf_cld
$ bash ./start_tcpdump.sh upf_mec
```

Latency test

Enter in the UE container:

```
$ b./enter_container.sh ue
```

Start ping test on the interfaces related to the two slices:

```
# ping -c 3 -n -I uesimtun0 10.45.0.1
```

```
vagrant@comnetsemu:~/comnetsemu/app/comnetsemu_5Gnet$ ./enter_container.sh ue
root@c5736900920c:/UERANSIM/build# ping -c 3 -n -I uesimtun0 10.45.0.1
PING 10.45.0.1 (10.45.0.1) from 10.45.0.2 uesimtun0: 56(84) bytes of data.
64 bytes from 10.45.0.1: icmp_seq=1 ttl=64 time=139 ms
64 bytes from 10.45.0.1: icmp_seq=2 ttl=64 time=141 ms
64 bytes from 10.45.0.1: icmp_seq=3 ttl=64 time=137 ms
```

```
ping -c 3 -n -I uesimtun1 10.46.0.1
```

```
root@c5736900920c:/UERANSIM/build# ping -c 3 -n -I uesimtun1 10.46.0.1
PING 10.46.0.1 (10.46.0.1) from 10.46.0.2 uesimtun1: 56(84) bytes of data.
64 bytes from 10.46.0.1: icmp_seq=1 ttl=64 time=35.0 ms
64 bytes from 10.46.0.1: icmp_seq=2 ttl=64 time=37.4 ms
64 bytes from 10.46.0.1: icmp_seq=3 ttl=64 time=37.5 ms
```

Bandwidth test

Enter in the UE container:

```
$ ./enter_container.sh ue
```

Start bandwidth test leveraging the two slices:

```
# iperf3 -c 10.45.0.1 -B 10.45.0.2 -t 5
```

```
--- 10.45.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 137.351/183.779/273.791/63.658 ms
root@c42c0b8d4823:/UERANSIM/build# iperf3 -c 10.45.0.1 -B 10.45.0.2 -t 5
Connecting to host 10.45.0.1, port 5201
[  5] local 10.45.0.2 port 52331 connected to 10.45.0.1 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  1.73 MBytes  14.5 Mbits/sec    3    312 KBytes
[  5]   1.00-2.00   sec   758 KBytes  6.22 Mbits/sec  350   51.3 KBytes
[  5]   2.00-3.00   sec  0.00 Bytes  0.00 bits/sec  100   47.4 KBytes
[  5]   3.00-4.00   sec  0.00 Bytes  0.00 bits/sec   41   36.9 KBytes
[  5]   4.00-5.00   sec   758 KBytes  6.23 Mbits/sec    3   30.3 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-5.00   sec  3.21 MBytes  5.38 Mbits/sec  497             sender
[  5]   0.00-5.13   sec  1.32 MBytes  2.16 Mbits/sec                  receiver

iperf Done.
root@c42c0b8d4823:/UERANSIM/build#
```

```
# iperf3 -c 10.46.0.1 -B 10.46.0.2 -t 5
```

```
iperf Done.
root@c42c0b8d4823:/UERANSIM/build# iperf3 -c 10.46.0.1 -B 10.46.0.2 -t 5
Connecting to host 10.46.0.1, port 5201
[  5] local 10.46.0.2 port 43823 connected to 10.46.0.1 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  3.32 MBytes  27.8 Mbits/sec  148   54.0 KBytes
[  5]   1.00-2.00   sec   885 KBytes  7.25 Mbits/sec   17   40.8 KBytes
[  5]   2.00-3.00   sec  1.30 MBytes  10.9 Mbits/sec    0   57.9 KBytes
[  5]   3.00-4.00   sec  1.30 MBytes  10.9 Mbits/sec    2   54.0 KBytes
[  5]   4.00-5.00   sec  1.30 MBytes  10.9 Mbits/sec   17   46.1 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-5.00   sec  8.07 MBytes  13.5 Mbits/sec  184             sender
[  5]   0.00-5.03   sec  7.10 MBytes  11.8 Mbits/sec                  receiver

iperf Done.
root@c42c0b8d4823:/UERANSIM/build#
```

Observe how the data-rate in the two cases follows the maximum data-rate specified for the two slices (2 Mbps for sst 1 and 10Mbps for sst 2).

`Now lets change in example2.py`

```
    with open( prj_folder + "/python_modules/subscriber_profile.json" , 'r') as f:
```

to

```
with open( prj_folder + "/python_modules/subscriber_profile_1.json" , 'r') as f:
```

save it

reboot your system and run your application again by using pervious steps this will configure.

Start again a bandwidth test in the UE

iperf3 -c 10.45.0.1 -B 10.45.0.2 -t 5

```
--- 10.46.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 35.681/37.017/39.186/1.547 ms
root@1405b93ad51d:/UERANSIM/build# iperf3 -c 10.45.0.1 -B 10.45.0.2 -t 5
Connecting to host 10.45.0.1, port 5201
[  5] local 10.45.0.2 port 41047 connected to 10.45.0.1 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.01   sec  2.03 MBytes  16.9 Mbits/sec    0    361 KBytes
[  5]   1.01-2.00   sec  2.59 MBytes  21.8 Mbits/sec  216    315 KBytes
[  5]   2.00-3.00   sec  1.97 MBytes  16.6 Mbits/sec  159    161 KBytes
[  5]   3.00-4.00   sec  1011 KBytes  8.28 Mbits/sec    0    176 KBytes
[  5]   4.00-5.00   sec  1011 KBytes  8.28 Mbits/sec   10    133 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-5.00   sec  8.57 MBytes  14.4 Mbits/sec  385             sender
[  5]   0.00-5.14   sec  6.38 MBytes  10.4 Mbits/sec                  receiver

iperf Done.
root@1405b93ad51d:/UERANSIM/build#
```

iperf3 -c 10.46.0.1 -B 10.46.0.2 -t 5

```
iperf Done.
root@1405b93ad51d:/UERANSIM/build# iperf3 -c 10.46.0.1 -B 10.46.0.2 -t 5
Connecting to host 10.46.0.1, port 5201
[  5] local 10.46.0.2 port 47427 connected to 10.46.0.1 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  5.55 MBytes  46.4 Mbits/sec    2    274 KBytes
[  5]   1.00-2.00   sec  2.04 MBytes  17.1 Mbits/sec   96   52.7 KBytes
[  5]   2.00-3.00   sec  1.85 MBytes  15.5 Mbits/sec    0   71.1 KBytes
[  5]   3.00-4.00   sec  1.85 MBytes  15.5 Mbits/sec    0   90.8 KBytes
[  5]   4.00-5.00   sec  3.09 MBytes  25.9 Mbits/sec    0    109 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-5.00   sec  14.4 MBytes  24.1 Mbits/sec   98             sender
[  5]   0.00-5.04   sec  12.8 MBytes  21.3 Mbits/sec                  receiver

iperf Done.
root@1405b93ad51d:/UERANSIM/build#
```

From the results you should observe that the achieved bit-rate have changed accordingly to the new setting.

1: what is mininet?

Mininet is an open-source network emulator that allows users to create virtual networks on a single machine. It uses lightweight virtualization technologies, such as Linux containers and network namespaces, to create a realistic network environment that can be used for testing, development, and education.

With Mininet, users can create complex network topologies that include switches, routers, hosts, and links, and then run actual networking software on them, such as OpenFlow controllers and switches, routing protocols, and network applications. This allows users to test and develop distributed networking applications in a controlled environment that accurately simulates real-world networking scenarios.

Mininet is widely used in academia and industry for research, education, and development purposes. It is particularly popular in the field of software-defined networking (SDN) as it allows researchers and developers to test and evaluate new SDN protocols and applications in a scalable, controlled, and reproducible environment.

Mininet is open-source software and can be downloaded and installed on Linux-based systems. It is actively maintained and developed by a community of contributors on GitHub.

Containernet framework?

Containernet is a Python-based network emulation framework that uses Docker containers to simulate network topologies. It is built on top of the Mininet network emulator and provides additional features such as support for multiple hosts, multiple controllers, and the ability to use Docker containers to emulate real-world devices. Containernet is often used to emulate complex network topologies and test network applications in a controlled environment. It also provides APIs for programmatically creating and manipulating network topologies, making it easy to integrate with other software tools.